PaX: Twelve Years of Securing Linux

PaX Team

LATINOWARE 2012.10.10

Linux

- Kernel developed by a community of companies and volunteers
- Started in 1991 by Linus Torvalds
- The 'brain' of the operating system (distributions, Android)
- Kernel vs. Userland

PaX

- Host Intrusion Prevention System (HIPS)
- Focus: exploits for memory corruption bugs
- Bugs vs. exploits vs. exploit techniques
- Threat model: arbitrary read/write access to memory
- Local/remote and userland/kernel
- Linux 2.2.x-2.4.x-2.6.x-3.x (2000-2012)
- Developed by the PaX Team :)
- grsecurity by Brad Spengler (spender)

PaX Features

- Runtime code generation control (non-executable pages)
- Address Space Layout Randomization (ASLR)
- Kernel self-protection
- Various infrastructure changes for supporting all the above

Software Development Lifecycle

- Idea/Design
- Development/Implementation
- Deployment/Configuration
- Operation/Maintenance

Vulnerabilities

- Conceptual/Design mistakes
  - e.g., lack of authentication or encryption
- Implementation mistakes
  - Common Weakness Enumeration
  - e.g., memory corruption bugs
- Deployment mistakes
  - e.g., wrong file permissions
- Operation mistakes
  - e.g., no monitoring/logging

Exploit Techniques

- Focus: exploits against memory corruption bugs
- Execute injected code (shellcode)
- Execute existing code out-of-(intended)-order (return-to-libc, ROP/JOP)
- Execute existing code in-(intended)-order (data-only attacks)

Design

- Prevent exploits from compromising userland applications
  - non-executable memory pages
  - runtime code generation control
  - address space layout randomization (ASLR)
  - control flow enforcement
  - (limited) data flow protection
- Prevent exploits from compromising the kernel itself
  - non-executable memory pages, etc

Implementation

- Static analysis (pay attention to clang/gcc/sparse warnings :)
- Runtime checks
  - userland/kernel separation
  - memory object lifetime checking
- Fuzzing (e.g., trinity by Dave Jones)

Deployment

- Mandatory Access Control (policies)
- Linux Security Modules (LSM)
  - Apparmor, SELinux, Smack, Tomoyo, Yama, etc
- grsecurity, RSBAC, etc
- Logging, log analysis
- Incident management

Overview

- Non-executable page support on i386 (PAGEEXEC/SEGMEXEC)
- Runtime code generation control (MPROTECT)
- Address Space Layout Randomization (ASLR, RANDEXEC)
- Compatibility (per-binary feature control, text relocations, trampoline emulation)

PAGEEXEC/SEGMEXEC/MPROTECT

- PAGEEXEC: paging based simulation of non-executable pages on i386 (in 2000, pre-NX days)
- SEGMEXEC: segmentation based simulation of non-executable pages on i386 (in 2002)
- MPROTECT: runtime code generation control (in 2000)
- NX-bit is in wide use nowadays (BSDs, iOS, Linux, Windows/DEP, etc)

ASLR

- Introduced in July 2001 as a stopgap measure (not how it turned out :)
- Idea: artificially inflated entropy in memory addresses (both code and data)
- Reduced exploit reliability
- In wide use nowadays (BSDs, iOS, Linux, Windows, etc)

Overview

- Non-executable kernel pages (KERNEXEC)
- Read-only kernel data (KERNEXEC, CONSTIFY)
- Userland/kernel address space separation (UDEREF)
- Restricted userland-kernel copying (USERCOPY)
- Instant free memory sanitization (SANITIZE)

KERNEXEC

- Non-executable pages for the kernel's address space
- Executable userland pages must not be executable from kernel mode
  - i386: code segment excludes the entire userland address space
  - amd64: compiler plugin or UDEREF
  - Supervisory Mode Execution Protection (`CR4.SMEP`) since Ivy Bridge (in mainline linux already)
- Page table cleanup: read-write vs. read-execute regions (kmaps)
- Special cases: boot/BIOS, ACPI, EFI, PNP, v8086 mode memory, vsyscall (amd64)

Constification

- Creates read-only data mappings
- Moves data into read-only mappings (`.rodata`, `.data..read_only`)
- Patches (descriptor tables, top level page tables, etc)
- Compiler plugin (ops structures)

UDEREF

- Prevents unintended userland access by kernel code
  - Disadvantage of the shared user/kernel address space
- i386: based on segmentation
  - data segment excludes the entire userland address space
- amd64: based on paging
  - remaps userland page tables as non-executable while in kernel mode
  - needs per-cpu page global directory (PGD)

USERCOPY

- Bounds checking for copying from kernel memory to userland (info leak) or vice versa (buffer overflow)
- spender's idea: `ksize` can determine the object's size from the object's address
- Originally heap (slab) buffers only
- Limited stack buffer support (see Future section)
- Disables SLUB merging

SANITIZE

- Reduces potential info leaks from kernel memory to userland
- Freed memory is cleared immediately
- Low-level page allocator, not slab layer
- Works on whole pages, not individual heap objects
- Kernel stacks on task death
- Anonymous userland mappings on munmap
- Anti-forensics vs. privacy

Overview

- gcc plugins (gcc 4.5-4.7)
- Kernel stack leak reduction (STACKLEAK)
- Function pointer structure constification (CONSTIFY)
- User/kernel address space separation for code only (KERNEXEC)
- Size parameter overflow detection&prevention (SIZE_OVERFLOW)

GCC plugins

- Loadable module system introduced in gcc 4.5
- Loaded early right after command line parsing
- No well defined API, all public symbols available for plugin use
- Typical (intended :) use: new IPA/GIMPLE/RTL passes

STACKLEAK plugin

- First plugin :)
- Reduces kernel stack information leaks
- Before a kernel/userland transition the used kernel stack part is cleared
- Stack depth is recorded in functions having a big enough stack frame
  - Sideeffect: finds all (potentially exploitable :) `alloca` calls
- Special paths for ptrace/auditing
- Problems: considerable overhead, races, leaks from a single syscall still possible

CONSTIFY plugin

- Automatic constification of ops structures (200+ in linux)
- Structures with function pointer members only
- Structures explicitly marked with a `do_const` attribute
- `no_const` attribute for special cases
- Local variables not allowed

KERNEXEC plugin

- Prevents executing userland code on amd64
- i386 achieves this already via segmentation
- Sets most significant bit in all function pointers
  - Userland addresses become non-canonical ones
- GIMPLE pass: C function pointers
- RTL pass: return values
- Special cases: assembly source, asm()
- Two methods: `bts` vs. `or` (reserves `%r10` for bitmask)
- Compatibility vs. performance

SIZE_OVERFLOW plugin

- Detects integer overflows in expressions used as a size parameter: `kmalloc(count * sizeof...)`
- Written by Emese Révfy
- Proper implementation of spender's old idea
- Initial set of functions/parameters marked by the `size_overflow` function attribute
- Walks use-def chains and duplicates statements using a double-wide integer type
- Special cases: asm(), function return values, constants (intentional overflows), memory references, etc
- More in our blog

Overview

- Control Flow Enforcement
- Size overflow detection & prevention

Control Flow Enforcement

- Compiler plugin
- (No) binary-only code support
- Assembly source instrumentation
- Runtime code generation support (Just-In-Time compiler engines)

Size Overflow Detection & Prevention

- Same plugin as used for the kernel
- Unique problems (build system integration, namespace collisions, etc)
- Already in progress (apache, glib, glibc, openssl, php, samba, syslog-ng, etc)
- Would have caught CVE-2012-2110 (ASN1 BIO vulnerability)
- Would have caught CVE-2012-2131 (the incorrect fix to 0.9.8v) too
- Needs support for c++ (chromium, firefox, etc)

Overview

- Link Time Optimization (LTO)
- LLVM/clang support
- Improved USERCOPY
- Improved REFCOUNT
- Improved STACKLEAK
- Control Flow Enforcement
- Limited data flow enforcement (KERNSEAL)
- PaX for hypervisors (HYPEREXEC)

LTO

- Mostly works with gcc 4.7
- Takes 5 minutes and 4GB RAM on a quad-core Sandy Bridge
- Problems: KALLSYMS, tracing, initcalls, section attributes
- Better support for other plugins (CONSTIFY, REFCOUNT, SIZE_OVERFLOW, STACKLEAK, USERCOPY)
- New plugins: static stack overflow checking, sparse attributes, etc

LLVM/clang

- http://llvm.org and http://clang.llvm.org
- Mostly works with linux-side patches only
- clang 3.1 and `-integrated-as`, `.code16gcc`/`.code16`
- `-fcatch-undefined-behavior` (ext4 triggers it on mount)
- LTO
- Port the gcc plugins to llvm
- New plugins for clang (not really feasible with gcc)

Improved USERCOPY

- Problem: `kmalloc-*` slabs
- Separate them into `kmalloc-user-*` and `kmalloc-*`
- Mark only `kmalloc-user-*` with `SLAB_USERCOPY`
- `kmalloc_user` vs. `kmalloc`
- Problem: find affected `kmalloc` calls
- Needs whole-tree static analysis (LTO plugin)

Improved REFCOUNT

- Problem: false positives (not every `atomic_t` variable is a reference counter)
- Statistical counters and unique identifiers (only increments), bitflags (directly set only)
- Needs whole-tree static analysis (LTO plugin) to find the above kind of variables

Improved STACKLEAK

- Problem: performance impact, races
- New per-task kernel stack used for USERCOPY
- Problem: find affected local variables
- Needs whole-tree static analysis (LTO plugin) to find the above kind of variables

Control Flow Enforcement

- Compiler plugin
- (No) support for binary-only modules
- Assembly source instrumentation
- Runtime code generation support?
- Performance impact is critical ($<$5% desired), very hard problem

KERNSEAL

- Ensures that certain kernel data cannot be modified unintentionally (arbitrary write bug)
- Credential structures, memory management data, filesystem metadata/data (page cache), etc
- Read-only slab
- Read-only kernel stacks (except for the current one :)
- Trusted pointer chains, trusted root pointers (current stack, per-cpu data?)

HYPEREXEC

- Virtualization does not increase security (despite marketing :)
- It introduces a new privilege level in the software stack (hypervisors)
- Hypervisors represent additional complexity and bugs
- Apply the kernel-self protection features to the hypervisor (Xen, KVM)
- Enforce (guest) kernel self-protection from a higher privilege level

http://pax.grsecurity.net

http://grsecurity.net

irc.oftc.net #pax #grsecurity